Explicit Modelling and Treatment of Repair in Prediction of Dependability

Jose Ignacio Aizpurua, *Member, IEEE*, Yiannis Papadopoulos, and Guillaume Merle

Abstract—In engineering practice, multiple repair actions are considered carefully by designers, and their success or failure defines further control actions and the evolution of the system state. Such treatment is not fully supported by the current state-of-the-art in dependability analysis. We propose a novel approach for explicit modelling and analysis of repairable systems, and describe an implementation, which builds on HiP-HOPS, a method and tool for model-based synthesis of dependability evaluation models. HiP-HOPS is augmented with Pandora, a temporal logic for the qualitative analysis of Temporal Fault Trees (TFTs), and capabilities for quantitative dependability analysis via Stochastic Activity Networks (SAN). Dependability prediction is achieved via explicit modelling of local failure and repair events in a system model and then by: (i) propagation of local effects through the model and synthesis of repair-aware TFTs for the system, (ii) qualitative analysis of TFTs that respects both failure and repair logic and (iii) quantification of dependability via translation of repair-aware TFTs into SAN. The approach provides insight into the effects of multiple and alternative failure and repair scenarios, and can thus be useful in reconfigurable systems that typically employ software to utilise functional redundancies in a variety of ways.

Index Terms—Repairable systems, dynamic dependability, reliability, reconfiguration.

Acronyms and Abbreviations

BDMP	Boolean-logic Driven Markov Process
BE	Basic Event
BTF	Base Temporal Form
CSQ	Cut-Sequence
DFT	Dynamic Fault Tree
DRBD	Dynamic Reliability Block Diagram
DSPN	Deterministic and Stochastic Petri Net
FA	Failure Automaton
FTA	Fault Tree Analysis
FMEA	Failure Mode Effects Analysis
GFT	Generalized Fault Tree
HiP-HOPS	Hierarchically Performed - Hazard Origin and Propagation Studies
H2SM	HiP-HOPS State Machine model
MCSQ	Minimal Cut-Sequence
PAND	Priority AND gate
PDF	Probability Density Function
POR	Priority OR gate
PS	Primary Standby
RBD	Reliability Block Diagram
SEFT	State Event Fault Tree
SAN	Stochastic Activity Network
SM	State Machine
TE	Top Event
TFT	Temporal Fault Tree

- *J. I. Aizpurua is with the Institute of Energy and Environment, University of Strathclyde, Glasgow, UK. E-mail: jose.aizpurua@strath.ac.uk*
- *Y. Papadopoulos is with the School of Engineering and Computer Science, University of Hull, Hull, UK. E-mail: y.i.papadopoulos@hull.ac.uk*
- *G. Merle is with the Sino-French Engineering School, Beihang University, Beijing, China. E-mail: guillaume.merle@gmail.com*

Manuscript received April 19, 2005; revised August 26, 2015.

1 Introduction

System dependability includes safety, reliability, availability, maintainability, confidentiality, and integrity attributes [1]. In this work we will consider reliability, availability, maintainability and safety. Traditionally, Reliability Block Diagrams (RBD) [2] or Fault Tree Analysis (FTA) [3] have been applied for the dependability analysis of systems. Whilst these methods can be used to assess the effects of combinations of faults, they are not able to capture system dynamics such as event sequences, triggering events or redundancies.

Industrial systems typically include repairable components either through self-healing or via external reconfiguration which repairs the failed component by restarting it or switching to an alternative component. In this paper we focus on dynamic repairable systems. The repair strategy determines the system's reaction to failures through repair events, e.g. when a component fails, the repair strategy determines which component should replace the failed component, and when the failed component is repaired whether it should come into operation or remain in the standby state.

There are dynamic dependability methods that deal with repair. Among these approaches we can distinguish between low-level pure stochastic models and high-level, analyst-friendly approaches. Pure stochastic models such as Markov Chains and Petri Nets can model any-complexity system including repairs. However, the system characterization is time-consuming, error-prone, and difficult to maintain due to the flatness and high number of states and events. High-level dynamic repairable approaches (e.g., Dynamic FTA [4], Dynamic RBD [5]) enable simpler and more

scalable specification of the dynamics including repair, but these approaches face limitations which will be discussed.

When assessing the dependability of dynamic repairable systems, both qualitative and quantitative assessments are useful [6]. For the dynamic qualitative analysis the concept of Minimal Cut-Sequences (MCSQs) was introduced [7]. MCSQ defines the minimal sequences of event failures which lead to top-event occurrence, i.e. the order in which some events occur determines the system failure occurrence. It is minimal in the sense that if any of the events of the sequence are removed, it is not possible to obtain another cut sequence leading to the TE occurrence. Although originally the concept was used with non-repairable Dynamic FTA [8], its use has been extended including the qualitative analysis of dynamic repairable systems [9]. In the latter, time-dependent events that cause system failure are investigated, and the designer can adopt design decisions to mitigate their effect e.g. via introduction of redundancies.

Currently, approaches treat failure and repair processes within their main failure modelling constructs, e.g. dynamic gates [4] or blocks [5]. For this, the system model with its repair logic must be mapped to the corresponding dependability analysis model counterpart. However, this model will not always work when the designer wishes to use repair as a design mechanism which allows alternative design decisions. In general, approaches to dependability analysis that account for repair do not provide mechanisms for explicitly considering different repair possibilities. Failed components are assumed functional following repair thus restoring their original function and position in the system configuration. The proposed approach differs from already existing approaches in that it allows explicit modelling of repair events as a part of the failure model. Using such events the designer may define multiple and alternative detailed repair scenarios and thus gain valuable insight into the impact of possible failure and repair sequences.

The proposed implementation of this approach is built around the event propagation concept of HiP-HOPS [10]. System components are augmented with failure and repair events and local effects. Using algorithms, local effects of failure and repair are then propagated among all the interconnected components. In the course of this, and using Pandora temporal operators [11], sequences of failure and repair that lead to hazards are captured in Temporal Fault Trees (TFTs). TFTs are analysed and MCSQs are determined. The disjunction of MCSQs that define the system failure behaviour forms the system structure function [12]. The resulting MCSQs show qualitatively the explicit influence of repair events on success and subsequent failure and give a causal view of system failure. Once the system's structure function is obtained, we introduce transformation algorithms that convert MCSQs to corresponding Stochastic Activity Networks (SAN) model [13]. At this point, the influence of the system's failure and repair can be predicted in a quantitative manner.

Therefore the contribution of this paper is a method which considers explicitly repair actions, enables improved qualitative dependability analysis of the combined effects of failure and repair, and quantifies the system failure probability via transformation of HiP-HOPS models into SAN. The explicit modelling and analysis of repair and its implementation are novel and can contribute to useful insights into the robustness of a design. In this paper, we assume reactive repair strategies, not prevention. The repair process is ideal, and therefore, the system and components are as-good-as-new after repair. Repair events for each component are independent, and two statistically independent events cannot occur simultaneously.

The paper is organised as follows. Section 2 presents related work for analysis of repairable systems. Section 3 presents an overview of the proposed approach. Section 4 focuses on qualitative analysis while Section 5 examines quantitative analysis and finally, Section 7 concludes.

2 Related Work

Many dynamic dependability models embed failure and repair processes in the modelling constructs to facilitate the system design and posterior dependability analysis.

Repairable Dynamic Fault Tree (DFT) [4] was developed to capture system dynamics through dynamic gates (Fig. 1) and embed failure and repair processes in the basic events (BEs) (Fig. 2a). Generalized Fault Tree (GFT) [14] adds repair processes to the DFT by adding an explicit repair-triggering mechanism which repairs a single or a group of BEs through predefined failure and repair processes.

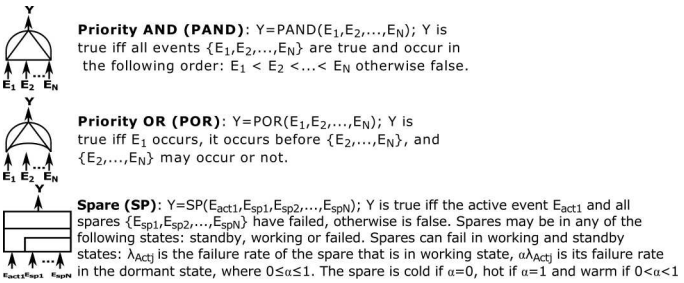


Fig. 1. A subset of Dynamic [4] and Temporal [11] Fault Tree gates.

Boolean logic Driven Markov Processes (BDMP) [15] add repair processes to the BEs which are triggered by Markov processes. That is, events or subsystems are triggered as a consequence of the occurrence of other events or subsystems. This mechanism provides flexibility to model the dependent behaviour of systems.

For BDMP and repairable DFT models, BEs are modelled with different predefined states depending if they are standby events or not (Figs. 2a, 2b). BEs are modelled as standby elements when they are inputs of a spare gate (DFT case) or when they are affected by the trigger mechanism (BDMP case). If the BE is a primary input of the spare or source of the trigger the initial state is the working (W) state. If the BE is a spare input or it is affected by the trigger, the initial state will be the standby state. In the standby state the BE can be in an operative (sW) or failed (sF) state.

Dynamic RBDs (DRBDs) [5] are based on the dynamic extension of RBDs. In DRBDs each block is modelled with three states (see Fig. 2c). The transitions between states are predefined and dependencies between components are also defined. The quantification of DRBD models can be done through Markov chains or Generalized Stochastic Petri Nets transforming the DRBD model into these formalisms. This allows dealing with repair but in a rather fixed set

of situations, and not in the general case considered in this paper, i.e. where multiple and alternative repair and reconfiguration scenarios are considered during design.

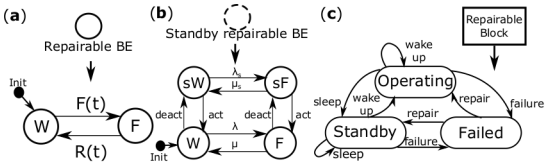


Fig. 2. (a) Repairable BE; (b) standby repairable BE; (c) DRBD states [5].

State-Event Fault Trees (SEFTs) add states and events to FTA for the analysis of dynamic systems [16]. SEFTs deal with functional and failure behaviour and allow the transformation into Deterministic and Stochastic Petri nets (DSPN) models (see Fig. 5 for an example).

DRBD, DFT or BDMP models provide flexibility to represent dynamic systems through dependencies, but this is limited when dealing with dynamic and repairable systems. Namely, standby situations are modelled correctly, but the specification of reconfiguration mechanisms requires more flexible modelling constructs (see next subsection). SEFTs are able to model reconfiguration more explicitly, however, the underlying dependability evaluation formalism is a flat DSPN model that includes all the transformations of states and gates. For complex systems this can cause state-explosion.

BDMP is a powerful mechanism, but only one type of dependency among components is allowed (the trigger can model two processes) and two triggers must not have the same destination component. BDMP has also been extended to address some of these limitations and model accurately complex systems through Switched Markov processes [17].

Among these models only BDMP and SEFT support qualitative analysis. However, due to the underlying assumptions, their qualitative analysis models do not fully cover classes of standby repairable systems. To the best of our knowledge, the qualitative analysis of dynamic repairable systems has not been fully addressed. Additionally, there is no support for compositional modelling of both failure and repair processes and the underlying stochastic model.

There are other high-level specification formalisms for dynamic repairable systems, but they are outside the scope of this work because they are focused on transforming a high-level design language into different dependability models. The reader is referred to [18] for an overview of high-level dependability specification formalisms and their transformation into different dependability formalisms.

To highlight the complications of repair in dependability analysis and illuminate the motivation of our work, we will introduce a repairable primary standby (PS) system. The system is simple enough for detailed analysis, but captures important dependability mechanisms encountered in safety-critical systems such as cold/warm redundancies, single points of failure, and reconfiguration strategies.

2.1 Running Example: Repairable PS System

The non-repairable version of PS in [19] has three main components (Fig. 3a): the primary component *A*; the monitoring component *S*, and the backup standby component *B*. *S* detects omission failures of *A* and it reconfigures the backup component *B*. Input *I* feeds *A* and *B* and is a common cause failure, while *OUT* must receive at least one input to perform correctly, otherwise the system fails. Fig. 3c shows the State Machine (SM) for the non-repairable PS system.

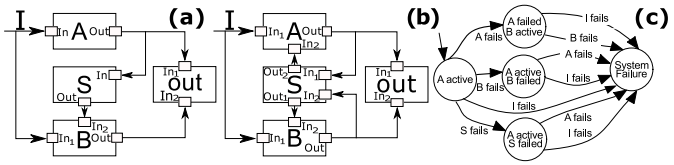


Fig. 3. PS system: (a) non-repairable configuration, (b) repairable configuration, (c) state machine of the non-repairable case.

For the dynamic qualitative analysis, Pandora and its temporal laws [11] are applied to the SM in Fig. 3c. For the comprehension of the example, it is sufficient to say that Pandora includes TFT gates and these are modelled with the next symbols. Logical OR "$+$", logical AND "$.$", and priority AND "$<$", where $A < B$ means that A must happen before B (see Section 4 for more details). To extract the structure function of the non-repairable PS system, we analyse the sequences of non-redundant events starting from the initial state *A active* that end in the *System Failure* state. We can see that the failure of *I*, or the failure of *A* and *B* in any order, or the failure of *S* prior to the failure of *A* lead to the system's failure. Translating these events into Pandora results in the structure function:

$$TE = F_I + F_A.F_B + F_S < F_A \quad (1)$$

where F_x designates the failure event of the component x.

The repairable PS system operates slightly differently. *S* detects the failure of *A* and activates *B* and also detects the failure of *B* and activates *A* (it is bidirectional, see Fig. 3b). In normal operation: input *I* (which is also repairable) feeds *A* and *B*. Initially *B* is in standby state and it needs a signal from *S* to activate. The system will operate safely so long as the *OUT* component receives a signal from *A* or *B*. *S* detects an omission failure of *A* and activates *B*. If *A* is repaired while *B* is operating, *A* remains in standby state until the failure of *B*. When *B* fails, *S* detects its failure and activates *A*.

S integrates fault detection and reconfiguration mechanisms. To detect omission failures, time- and value-based fault detection mechanisms can be used. In order to reactivate *A* and *B* from the standby state, these components have an internal mechanism which is triggered by the reconfiguration signal issued by *S*. Thereby, *S* generates trigger signals according to the system state.

2.2 Dynamic Qualitative Analysis: Failure Automaton

The Failure Automaton (FA) defines the effect of all possible failure and repair events [20]. To define the FA, first it is necessary to model the system failure logic using a dependability model. Fig. 4a shows the DFT model of the repairable PS system and Fig. 4b shows the equivalent BDMP model which uses the trigger mechanism (dashed arrow) to model dependencies and reactivate spare components.

DRBD and SEFT models enable modelling reconfiguration strategies. Fig. 5 shows the SEFT model of the PS

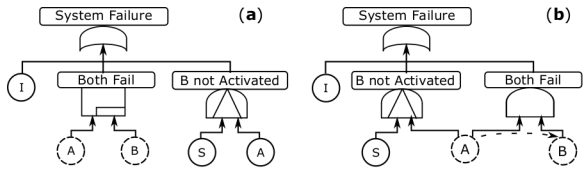


Fig. 4. Failure models of the repairable PS system: (a) DFT (b) BDMP.

system. The model of B is the same as A adding another output port from the *B_KO* state, and the model of I is the same as S. The only difference between *A-B* and *I-S* is that each of these models will have their own failure and repair rates. We explicitly model the reconfiguration behaviour of S by adding the conditions in which the components A and B should be reactivated (*Reactivate_A*, *Reactivate_B*).

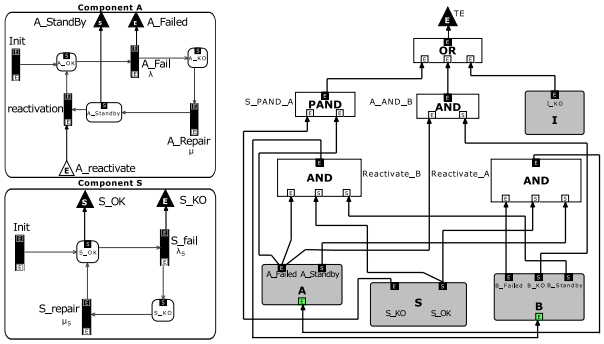
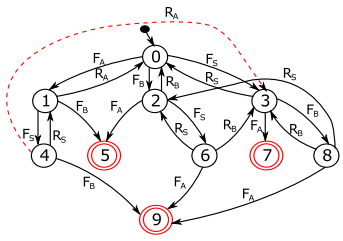


Fig. 5. SEFT model of the repairable PS system.

For brevity we will limit the discussion to the SEFT model, but note that the DRBD model results in the same qualitative and quantitative dependability results as SEFT.

The qualitative analysis of the reviewed models results in the FA shown in Fig. 6. Note that the unsafe states are modelled as absorbing states (double circles). For simplicity input I is not included. It is a single point of failure in disjunction with other MCSQs in the structure function and the effect is obvious.



System State	Components State		
	A	B	S
0	A_W	B_S	S_W
1	A_F	B_W	S_W
2	A_W	B_F	S_W
3	A_W	B_S	S_F
4	A_F	B_W	S_F
5	A_F	B_F	S_W
6	A_W	B_F	S_F
7	A_F	B_S	S_F
8	A_W	B_F	S_F
9	A_F	B_F	S_F

Fig. 6. FA of the repairable PS system using state-of-the-art approaches.

The FA of Fig. 6 shows that the reviewed approaches will struggle to capture accurately the dynamics of the repairable PS system. We list a few subtle issues. (i) The transition from the state #4 to #3 (dashed red line) should not be allowed, because S is failed. In the case of the sequence F_A<F_S<R_A, the system should continue operating with B because the component that enables the trigger function (S) is failed and A cannot be restored to operative state. (ii) The logic of the spare gate integrated in all these approaches (except DRBD and SEFT) is not sufficient rich to capture the standby behaviour of the repairable PS system. A is initially working and when it fails S activates B. Once A is repaired it should remain in standby state until B fails. However, in the methods applied here, A is activated as soon as its repair happens and B returns to the standby state. The subtlety of the design cannot be expressed. (iii) There is an additional MCSQ which can cause system failure post-repair which is also not captured (see Section 4 for details).

Note that the repair of A and B is independent of the operation of S. That is, they can be repaired even if S has already failed, and their default state after repair is standby. However, the reactivation of these components is dependent on the reconfiguration signal issued by S. These are issues that cannot be easily modelled in the state-of-the-art.

Embedding repair within the failure logic using fixed constructs such as specialised gates indeed limits flexibility and causes partial loss of insight into the effects of repair. If the designer adopts the simple decision of changing the behaviour of BEs from non-repairable to repairable, the system will fail to cover all the possible failure situations.

In our view, unless repair is modelled explicitly and qualitative analysis that includes failure and repair events is performed, the designer may fail to address all failure scenarios. That is at the heart of our proposal and distinguishes it from other work in the field.

3 Overview of Approach

HiP-HOPS defines a modelling approach for system failure annotation and posterior processing algorithms that extract FTA models from the system topology [10]. In classical HiP-HOPS, the local failure logic of each component in the system architecture is defined. The local failure logic defines how each component reacts to incoming failures and how the component fails by itself. This is expressed as a set of failure expressions which relate the component's output failures to input and internal failures.

After the model has been augmented, algorithms traversing the system topology determine how the local failures propagate through connections in the model and cause the failures at the output of the system. This is captured deductively (from effects towards causes) in a set of interconnected FTs which share branches and BEs that arise from the dependencies in the system topology. The graph of connected FTs is post-processed to calculate system dependability.

HiP-HOPS failure expressions may include temporal operators in Pandora logic [11]. Pandora operators allow specifying sequence-dependent and relative temporal ordered events and thus modelling of dynamic systems. Use of Pandora in HiP-HOPS means creation of system TFTs.

In this paper, we extend HiP-HOPS with repair processes. The failure and repair behaviour of each component is described locally using input and internal failure and repair events. Furthermore, we integrate component State Machine (SM) models within HiP-HOPS to gain better understanding of behaviour and provide flexibility. For the qualitative analysis, SM models are extended into FA models to include all the failure causes and effects of a component. Thereby, the system design specification is created by

separating the component interface and detailed behaviour. Fig. 7 shows an overview of the approach.

The *system architecture specification* (#1) is comprised of a set of interconnected components that exchange material, energy or data flows. There are failure events and repair events to activate reconfiguration mechanisms, or status information to monitor the system performance. For each component, the *component specification* (#2) is performed in two parallel activities. The *interface specification* (#3) defines, for each component, the I/O interfaces relating input and internal event occurrences (both failure and repair events) to the output events. The detailed specification determines how the events are propagated by each component. Firstly, the *functional specification* (#4) is defined, which models the minimal and sufficient set of states, events, and propagated functional events by means of interconnected SMs. Then, augmenting the functional specification, the component *Failure Automaton* (#5) is specified introducing the effect of all influencing events.

Note that the SM used for the functional specification and the FA used for qualitative analysis use the same modelling constructs with two differences. The FA has marked states to identify the failure states and the functional specification is a subset of the FA. That is, the FA integrates all failure causes and effects, and the SM includes minimal necessary set of states and propagated events for operation.

The *synthesized component TFT* (#6) is extracted from the FA of each component as a disjunction of combinations and sequences of events that define the possible failure causes. Subsequently, *annotations* (#7) are performed for each component including input failures, internal failures, repair events, and output propagated failure modes. Once all components have been annotated, failure and repair events are propagated through their interfaces defining the overall behaviour of the system. Thereby, the *System Failure Automaton* (#8) is extracted from the system architecture specification and component annotations. The system architecture can be seen as a set of repairable TFTs, in which each component specifies its part in the synthesis of the global repairable TFT from this set.

The *TFT synthesis* (#9) step creates temporal expressions of system failure which contain component failure and repair events. These expressions may include redundant terms. Using the reduction rules in [11] the *TFT minimization* (#10) activity is performed. Redundant terms are removed and structure function is produced, which contains only non-redundant MCSQ sets. To automate the *TFT synthesis* activity, we propose an algorithm to generate TFTs from the SMs.

The *model transformation* (#11) activity is where quantitative evaluation takes place. It takes *interconnected component state machines* (denoted *Behavioural Model*) and the structure function of the system (denoted *Failure Model*) as input, and outputs *Stochastic Activity Networks (SAN)* [13]. These are used for *quantification* (#12) of the system failure probability for any PDF of failure and repair events, e.g. [21]. In Sections 4 and 5 we focus on qualitative analysis, i.e. calculation of minimal structure function, and quantitative analysis, i.e. calculation of system failure probability, respectively.

4 Qualitative Analysis

For qualitative analysis we use Pandora which introduces TFT gates (ordered from lower to higher priority) [11]: *Priority AND* (PAND, symbol "<") Y=In_1<In_2, Y occurs if In_1 occurs before In_2; *Priority OR* (POR, symbol "|") Y=$In_1|In_2$, Y occurs if In_1 occurs before or without the occurrence of In_2; and *Simultaneous AND* (SAND, symbol "&") Y=In_1&In_2, Y occurs if In_1 occurs at the exact time as In_2.

Pandora introduces temporal laws that can be used for minimization of Pandora expressions. Temporal laws are validated through temporal truth tables (TTTs), which are equivalent to Boolean truth tables. Temporal laws aid in this process and provide mechanisms for removing redundant sequences (e.g., Priority Laws) and contradictions that may exist in temporal expressions (e.g., Mutual Exclusion Laws). Table 1 displays some examples of Pandora Temporal Laws and temporal equivalences derived from TTTs.

TABLE 1
Examples of Pandora Temporal Laws.

ID	Temporal law	Example
1	Absorption	$X.(X<Y) \Leftrightarrow (X<Y)$
2	Pand right distr.	$(X.Y)<Z \Leftrightarrow (X<Z).(Y<Z)$
3	Pand: Not Left Distributive	$X<(Y.Z) \not\Leftrightarrow (X<Y).(X<Z)$; $X<(Y.Z) \Leftrightarrow Y.(X<Z)+Z.(X<Y)$
4	Conj. Complet.	$X.Y \Leftrightarrow X<Y+X\&Y+Y<X$
5	Extension	$(X<Y).(Y<Z) \Leftrightarrow (X<Y).(X<Z).(Y<Z)$
6	Temp. Equiv.	$X<(Y\|Z) \Leftrightarrow (X<Y).(Y\|Z)$

For complete details about Pandora and temporal laws please refer to [11]. We use Pandora's TFT gates and temporal laws to express the system's failure and repair behaviour, and extract the non-redundant causes that lead the system to failure, i.e. structure function.

4.1 Representing Repairable Systems with Pandora

The goal of the qualitative analysis is to identify the necessary and sufficient causes that lead the system to failure. The minimization of the structure function is crucial to remove redundant sequences and loops.

Retaking the FA shown in Fig. 6, we identify each path i, denoted by π_i, that the system can take from the initial state S_0 to the failure state S_5, S_7, or S_9. Each path π_i consists of a set of ordered states, S_i, from which the system starts, passes through intermediate states and ends in a non-recoverable failure state π_i=$\{S_0, S_1, \ldots, S_{\text{failure-1}}, S_{\text{failure}}\}$. Considering only the failure state #5 in Fig. 6, the set of possible paths that lead the system to a failure are: π_1=$\{S_0, S_1, S_5\}$, π_2=$\{S_0, S_2, S_5\}$, π_3=$\{S_0, S_3, S_8, S_2, S_5\}$, π_4=$\{S_0, S_2, S_6, S_3, S_8, S_2, S_5\}$, π_5=$\{S_0, S_1, S_0, S_1, \ldots, S_0, S_1, S_5\}$, π_6=$\{S_0, S_2, S_0, S_2, \ldots, S_0, S_2, S_5\}$, π_7=$\{S_0, S_3, S_0, S_3, \ldots, S_0, S_3, S_8, S_3, S_8, \ldots S_8, S_2, S_0, S_2, \ldots S_0, S_2, S_5\}$, π_8=$\{S_0, \ldots, S_5\}$.

From all the infinite paths that end in the failure state, an adaptation is necessary to evaluate the finite set of scenarios leading the system to failure. For qualitative analysis, only the paths π_1 and π_2 provide useful information identifying the original system failure causes. The remaining paths do not include new information and they can be determined from π_1 or π_2. For instance, π_3 includes the path π_2 plus the failure and repair events of S. Accordingly, the paths which

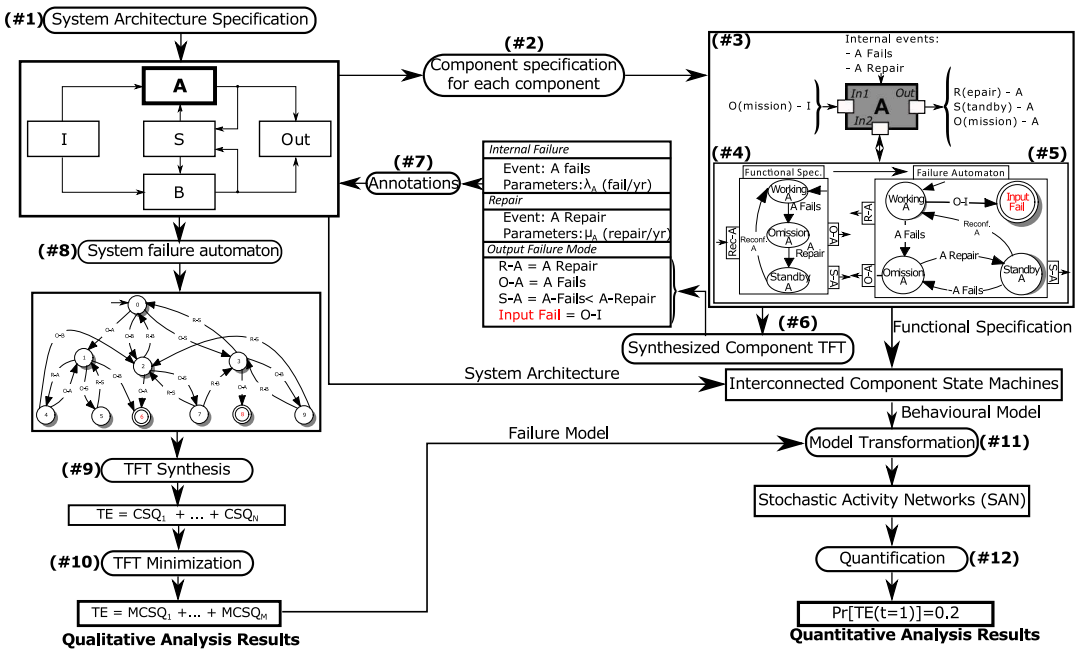


Fig. 7. Overview of the repairable HiP-HOPS approach.

include failure-repair loops ($\pi_4 - \pi_7$) do not provide information about new causes, but only possible failure/repair iterations until the component moves to the system failure state. For simplicity we have only considered the failure state #5, but #7 and #9 are also failure states and the same qualitative analysis process should be repeated.

4.2 Qualitative Analysis: state-of-the-art approaches

From the FA of the reviewed approaches (Fig. 6), we identify these non-redundant paths that cause system failure: π_1={S_0, S_1, S_5}, π_2={S_0, S_2, S_5}, π_3={S_0, S_3, S_7}, π_4={S_0, S_1, S_4, S_9}, π_5={S_0, S_2, S_6, S_9}, π_6={S_0, S_3, S_8, S_9}.

By applying Pandora's laws to these sequences of states and include the influence of the component I, the following minimal cut sequence sets are extracted:

$$
\begin{aligned}
MCSQ_0 &= F_I \\
MCSQ_1 &= (F_A < F_B).(F_A|F_S).(F_B|R_A) \\
MCSQ_2 &= (F_B < F_A).(F_B|F_S).(F_A|R_B) \\
MCSQ_3 &= (F_S < F_A).(F_S|F_B).(F_A|R_S) \\
MCSQ_4 &= (F_A < F_S).(F_A|F_B).(F_S < F_B).(F_S|R_A).(F_B|R_S) \\
MCSQ_5 &= (F_B < F_S).(F_B|F_A).(F_S < F_A).(F_S|R_B).(F_A|R_S) \\
MCSQ_6 &= (F_S < F_B).(F_S|F_A).(F_B < F_A).(F_B|R_S).(F_A|R_B)
\end{aligned} \tag{2}
$$

From Eqs. in (2) we can see that the sequences of events are expressed using the PAND gate, and constraints which involve priorities between events are determined using the POR gate. For instance, in $MCSQ_1$: (i) A must fail prior to the failure of B, (ii) if S fails, it must fail after the failure of A, and (iii) if A is repaired, it must be repaired after B fails.

4.3 Qualitative Analysis: Repairable HiP-HOPS

To perform the qualitative analysis we first introduce the independent failure and repair operation of each component, i.e. functional specification (Fig. 7). The functional specification of each component includes a SM (later treated as a FA adding further events and states) and HiP-HOPS propagation interfaces. For generality, let us integrate these modelling constructs in the definition of HiP-HOPS SM:

Definition 1: HiP-HOPS State Machine - A HiP-HOPS State Machine (H2SM) is a 6-tuple $H2SM =< S, \sum, \delta, s_0, EP_{In}, EP_{Out} >$ where:

- $S \neq \emptyset$ is a set of states.
- $\sum$ is a set of events. Each event $e \in \sum$ is of the type τ={*immediate*, *stochastic*, *conditional*}, where *stochastic* events occur after a random period of time specified by their corresponding PDF, *immediate* events occur instantaneously, and *conditional* events are of the form $e = event[guard(s)]|action$. That is, if the $guard(s)$ condition(s) holds when the *event* occurs, the *action* event is executed.
- δ: $S \times \sum \rightarrow S$ is a transition function such that for $(u, u') \in S^2$ and $e \in \sum$, $u' = \sigma(u, e)$ iff e is incident from u to u', which is written as $u \xrightarrow{e} u'$.
- $s_0 \in S$ is the initial state.
- EP_{In} is the set of incoming events $e_{EP_{In}} \in EP_{In}$ propagated to the $H2SM$ model by other $H2SM$ components.
- EP_{Out} is the set of outgoing events $e_{EP_{Out}} \in EP_{Out}$ propagated by the $H2SM$ model to other components.

The functional specification of each component is given in a HiP-HOPS SM. Fig. 8 shows the repairable PS components and their functional interfaces. For instance, the component A is defined as follows: *S*={*A Working*, *A Failed*, *A Standby*}; $\sum$={*A Fails*, *A Repair*, *Reconfigure A*}, where *A Fails* and *A Repair* are stochastic and *Reconfigure A* is *immediate*; δ is defined in Fig. 8a; s_0 = *A Working*; EP_{In} = {Rec-A}; EP_{Out} = {O-A, S-A}.

Note that the I/O propagated events are described using the HiP-HOPS notation, i.e., [event class]-[component]. In this case: O(mission)-A; S(tandby)-A; and Rec(onfigure)-A.

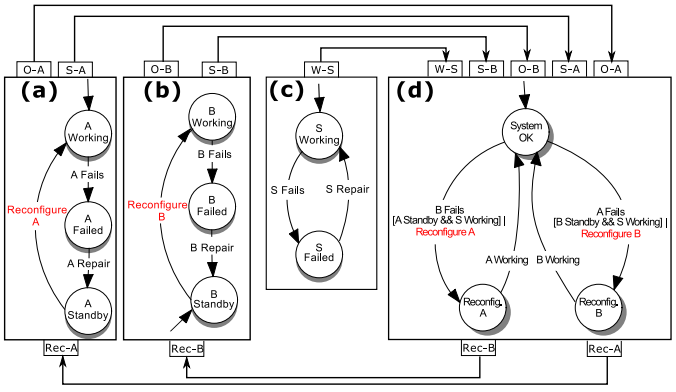


Fig. 8. Functional specification of the repairable PS system components.

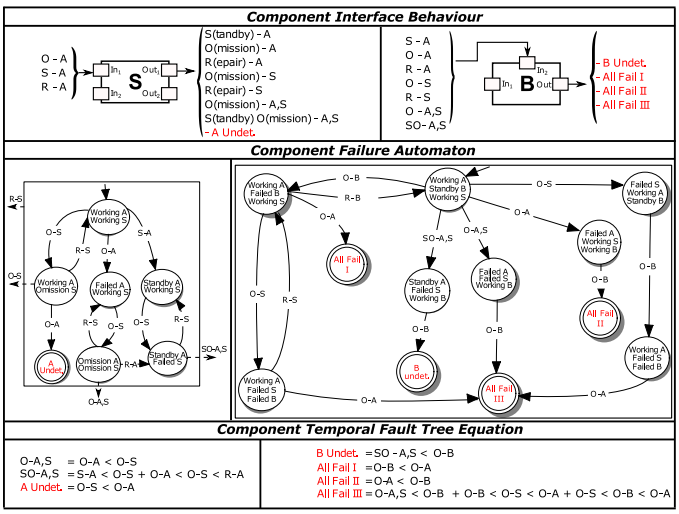


Fig. 9. Characterization of the components B and S.

To describe the stochastic failure and repair events of each component, their failure and repair PDFs are used. S (cf. Fig.8d) sends an immediate signal (*Rec-A, Rec-B*) to activate (or reconfigure) A or B when required. Unless S is working, it is not possible to activate A and B.

The propagation of the failure and repair events between the components enables an accurate and manageable specification of the system operation. However, when performing the qualitative analysis of the system, additional failure and repair events need to be propagated to evaluate all the possible causes/effects and their influence on the system performance. These states/events are formalised in the FA.

For clarity, the following decisions have been adopted when modelling the FA of components: the propagation of the failure and repair events is done in one direction (from component A to B) and once a component reaches an absorbing failure state, it is not further propagated because it causes the system failure (e.g., *Input Fail* event in Fig. 7). If we consider events in both directions the component FA will have overlapping information. For instance, if we include the influence of S and B in the FA of A via interface In_2 (see Fig. 3b), we will end up with the same component FA as if we consider the influence of A and S in the component FA of B. The characterization of each component with their interface behaviour, FA and corresponding TFT equation are shown in Fig. 7 (component A) and Fig. 9 (S and B).

The system FA is constructed as shown in Fig. 10 based on the component FA of components A, B, and S. For simplicity I is not included. In Fig. 10 we can see that the system cannot move from the state #1 back to #0 because if A is repaired after its failure, B would be working and A will not come back into operation until B fails. The possible states are displayed in Table 2.

4.4 Representing the HiP-HOPS FA using Pandora

We can extract Cut-Sequences (CSQ) directly from the FA in Fig. 10. We will focus on the paths that end in the failure states #6, #8, and #12 because the path to #11 contains non-minimal cut sequences as it includes the situation in which all the events have failed. Namely, a CSQ_i is included in one of the CSQ_j if it satisfies the criterion [12]: $CSQ_i . \sum_{j \neq 1} CSQ_j = CSQ_i$. That is, if CSQ_i can be included in one of the CSQ_j, it is redundant and can be removed from the structure function.

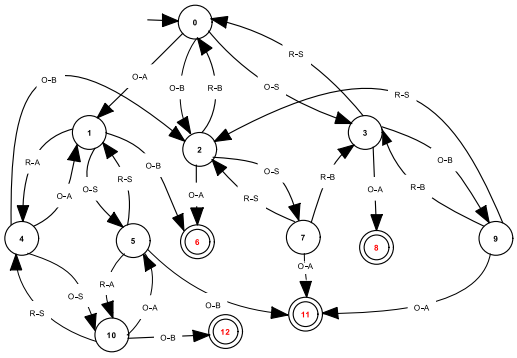


Fig. 10. Failure Automaton of the repairable PS system.

TABLE 2
States of components in Fig. 10.

System State	Components State		
	A	B	S
0	A_W	B_S	S_W
1	A_F	B_W	S_W
2	A_W	B_F	S_W
3	A_W	B_S	S_F
4	A_S	B_W	S_W
5	A_F	B_W	S_F
6	A_F	B_F	S_W
7	A_W	B_F	S_F
8	A_F	B_S	S_F
9	A_W	B_F	S_F
10	A_S	B_W	S_F
11	A_F	B_F	S_F
12	A_S	B_F	S_F

These are the failure paths: π_1={S_0, S_1, S_6}, π_2={S_0, S_2, S_6}, π_3={S_0, S_3, S_8}, π_4={S_0, S_1, S_4, S_{10}, S_{12}}, π_5={S_0, S_1, S_5, S_{10}, S_{12}}. If we apply Pandora's TFT gates to these paths we end up with the following equations:

$$
\begin{aligned}
&CSQ_O = F_I \\
&CSQ_1 = (F_A < F_B).(F_A|F_S).(F_B|R_A) = (F_A < (F_B|R_A)).(F_A|F_S) \\
&CSQ_2 = (F_B < F_A).(F_B|F_S).(F_A|R_B) = (F_B < (F_A|R_B)).(F_B|F_S) \\
&CSQ_3 = (F_S < F_A).(F_S|F_B).(F_A|R_S) = (F_S < (F_A|R_S)).(F_S|F_B) \\
&CSQ_4 = (F_A < R_A).(R_A < F_S).(F_S < F_B).(F_B|R_S) \\
&CSQ_5 = (F_A < F_S).(F_S < R_A).(R_A < F_B).(F_B|R_S)
\end{aligned} \tag{3}
$$

The POR gate enables identifying consistently each failure path avoiding non-deterministic states. We remove the redundancies existing in the CSQs to get the MCSQs (cf. Fig. 7). Namely, we can apply Pandora's Conjunctive Completion Law to CSQ_4 and CSQ_5 (Table 1 ID4). Eq. (4) shows the MCSQ set of the PS system including repair events.

$$
\begin{aligned}
TE = F_I + (F_A < (F_B|R_A))(F_A|F_S) + (F_B < (F_A|R_B))(F_B|F_S) \\
+ (F_S < (F_A|R_S))(F_S|F_B) + [F_A < (R_A.F_S)] < (F_B|R_S)
\end{aligned} \tag{4}
$$

The minimal cut sequences CSQ_1, CSQ_2, and CSQ_3 have been identified by the application of previous approaches [Eqs. in (2)], but the next MCSQ expression has not been identified: $MCSQ_4 = [F_A < (R_A.F_S)] < (F_B|R_S)$, which means that the following sequence of events have to occur to cause the TE occurrence: initially A fails and as a result, S activates B. While B is operating, two events must happen (in any order): repair of A and failure of S. At this instant: B is operating, A is in standby state, and S is failed. Finally B fails and the repair of S happens (if it occurs at all) after the failure of B. Therefore, A cannot be reactivated.

Eq. (4) defines the system failure as function of failure and repair events. Fig. 11 shows the corresponding failure model including explicit repair events (cf. symbols in Fig. 1).

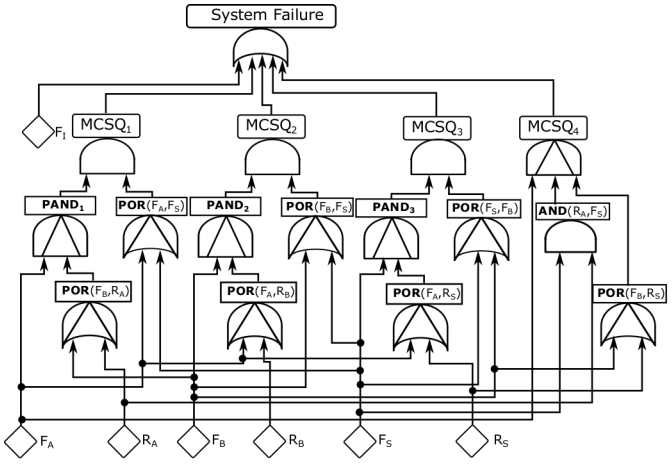


Fig. 11. Pandora TFT model of the repairable PS system.

Failure and repair events are modelled as undeveloped events (rhomboid symbol) because the behaviour of these events will be determined by the functional operation of each component.

4.5 Qualitative Analysis Discussion

If we compare the structure function in Eq. (4) extracted from the FA of the proposed approach (Fig. 10) with the equations extracted from the FA of other approaches applied on the same system (Fig. 6), we see uncovered failure situations in the latter. (i) In Fig. 10 the transition from state #1 to #0 is not modelled, while in Fig. 6 it is modelled. In the repairable PS system, once A fails it will remain in standby state until B fails and S is operative. (ii) In Fig. 6 the transition from state #4 to #3 is not possible because S must be operative in order to reconfigure A. This situation is correctly described in Fig. 10 with the transition from state #5 to #10. (iii) The failure event *B undet.* (Fig. 9, component B) included in the state #12 of the FA of the proposed approach (cf. Fig. 10) is not included in the FA of the reviewed approaches.

A key difference made by the use of Pandora, which allows rich modelling and analysis of failure scenarios that include repair events. For instance, reactivation events that must occur to prevent the occurrence of the system failure could be defined with POR gates in failure equations and their effects properly captured at system level.

5 Quantitative Analysis

The goal of the quantitative analysis is to evaluate the system failure probability with the possibility to model any PDF for failure and repair events. Failure events are stochastic events which are not dependent on other operational conditions. However, repair events occur as a result of other actions, and they need to be considered accordingly including the conditions to initiate the repair and the repair event itself. Therefore, specifying directly the events in Eq. (4) with the corresponding failure and repair PDF and evaluating the top-event failure probability is infeasible because the specification of the PDF of the repair events is not trivial.

5.1 Overview of the Quantitative Analysis

Fig. 12 provides an overview of the quantification process. Firstly the designer models the system in HiP-HOPS and extracts the *system structure function* according to the *dynamic qualitative analysis* process described in Section 4. Secondly, the *transformation* of the functional specification of the interconnected system components into the interconnected SAN component models is performed creating the *behavioural model*.

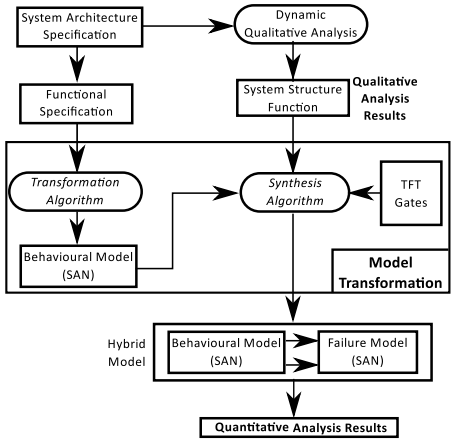


Fig. 12. Quantitative analysis of the repairable HiP-HOPS approach.

Finally, the *system structure function* is implemented in SAN by means of the *TFT gates*. As a result of this process, the structure function is connected with the behavioural

model in order to quantify the system failure probability. The resulting hybrid model connects the behavioural and failure model by implementing HiP-HOPS event propagation concepts.

The transformations from the source (H2SM and Pandora) into target (SAN) models are automated by implementing: (i) a transformation algorithm to convert the functional specification of interconnected system components specified as H2SM (see Definition 1) into their counterpart SAN component models, i.e., the behavioural model. (ii) A synthesis algorithm to compile automatically any structure function expressed with Pandora into SAN models, i.e. the failure model. To this end, the interfaces of the behavioural model are linked with the events in the failure model according to the system structure function.

5.2 Preliminaries on Stochastic Activity Networks

SAN [13] provides the modelling capabilities and solver support to meet the requirements for modelling complex repairable systems effectively including (i) the specification of system dynamics (time-dependent events, sequential events, standby situations), (ii) the modelling of repair processes, (iii) the support for the specification of any PDF of failure and repair events, and (iv) the support for the modular construction of the failure and repair model and the underlying stochastic analysis model. Refer to [22] for more details about the Möbius implementation framework.

SAN extends stochastic Petri Nets generalizing the stochastic relationships and adding mechanisms to construct hierarchical models [13]. Accordingly, SAN can overcome some of the limitations of pure stochastic models. Figure 13 shows SAN modelling primitives.

Fig. 13. SAN modelling mechanisms.

Before formally defining a SAN model, let us define concepts related with the marking of the net. Places represent the state of the modeled system. Each place contains a certain number of tokens defining the marking of the place. Let P denote the set of places of the network. If S is a set of places ($S \subseteq P$), a marking of S is a mapping $\mu : S \to \mathbb{N}$. We will denote interchangeably the marking function of the place x as $m(x)$ or $\mu(x)$, e.g., $\mu(x) = m(x) = 1$ means that the place x has a marking equal to 1. Similarly, the set of possible markings of S is the set of functions $M_S = \{\mu | \mu : S \to \mathbb{N}\}$.

Activities fire based on the conditions defined over the marking of the net and their effect is to modify the marking (μ) of the places. The completion of an activity of any kind is enabled by a particular marking of a set of places. The presence of at least one token in each input place enables the firing of the activity removing the token from its input places and placing them in the output places. Formally a SAN model is defined as follows [13]:

Definition 2: Stochastic Activity Networks (SAN) - A SAN model is a 5-tuple $SAN =< AN, \mu_0, C, F, G >$ where:

- AN is the activity network, which is a 8-tuple $AN =< P, A, IG, OG, \gamma, \tau, i, o >$, where $P \neq \emptyset$ is some finite set of places, $A \neq \emptyset$ is a finite set of input places, $IG \neq \emptyset$ is a finite set of input gates, each input gate $ig \in IG$ defined as a triple $ig =< G, ena, f >$ where $G \subseteq P$ is the set of places associated with the gate, $ena : M_G \to \{0, 1\}$ is the enabling predicate of the gate, and $f : M_G \to M_G$ is the input function of the gate, and OG is a finite set of output gates, each output gate $og \in OG$ defined as a pair $og =< G, f >$. Furthermore, $\gamma : A \to \mathbb{N}^*$ specifies the number of cases for each activity, and $\tau : A \to \{timed, instantaneous\}$ specifies the type of each activity. The net structure is specified via the functions i and $o : i : IG \to A$ maps input gates to activities, while $o : OG \to \{(a, c) | a, c \in [\![1, \gamma(a)]\!]\}$ maps output gates to cases of activities.
- $\mu_0 \in M_P$ is the initial marking.
- C is the case distribution assignment.
- F is the activity time distribution function assignment, an assignment of continuous functions to timed activities such that for any timed activity a, $F_a : M_P \times \mathbb{R} \to [0, 1]$. Furthermore, for any stable marking $\mu \in M_P$ and timed activity a that is enabled in μ, $F_a(\mu, .)$ is a continuous PDF called the activity time distribution function of a in μ; $F_a(\mu, \tau) = 0$ if $\tau \leqslant 0$.
- G is the reactivation function assignment.

In the proposed approach we do not use output gates and reactivation functions. Furthermore, we assume that the number of cases for each activity is one, simplifying the final structure of the SAN model. Accordingly, for clarity we have avoided introducing the complete definitions of the case distribution assignments and reactivation function assignments (for a complete specification refer to [13]).

The SAN models which include the specified SAN elements (Fig. 13) are modelled in an atomic model. The join operator links through a compositional tree structure different SAN models in a unique composed model. It is possible to link atomic models, composed models, or combinations thereof. In the tree structure, the composed/atomic SAN models are linked through join operators using the shared places between the composed/atomic SAN models. Thus, the analyst can focus on specific characteristics of the system behaviour through fit-for-purpose atomic/composed models and later join independently validated models to obtain a more complex composed system model (e.g. see [23]).

The performance measurements are carried out through *reward functions* defined over the designed model. Reward functions are evaluated as the expected value of the reward function and they are defined based on the marking of the net (*state reward function*), e.g. quantification of the probability for being in a specific place, and completion of activities (*impulse reward function*), e.g. count the number of times an activity triggers within a time interval.

Fig. 14 shows a simple repairable component example. In this case the SAN places are initialized to working state <m(`W`)=1, m(`F`=0)>. The token will move from `W` to the `F` place according to the cumulative distribution function (CDF) determined by the `fault` timed activity. The time to failure is calculated with the parameters of the `fault` activity and after the time to failure has elapsed the system

moves to the failed state <m(W)=0, m(F=1)>. After moving to the failure state the time to repair is calculated from the `repair` timed distribution and the token moves from `F` to the `W` place after the calculated time to repair has elapsed.

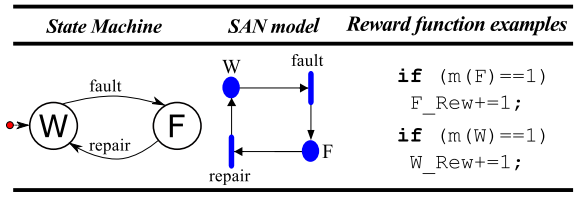


Fig. 14. Repairable asset example in SAN.

We focus on Monte Carlo simulations for the quantification of different probabilities. For evaluating the failure probability or availability we use the reward functions indicated in Figure 14 with `F_Rew` and `W_Rew` reward variables respectively. Formally, for evaluating the probability of a generic place x, at time instant t, first we define the reward function, $r_x(t)$:

$$r_x(t) = \begin{cases} r_x(t)+1 & \text{if} \quad m(x)=1 \\ r_x(t) & \text{if} \quad m(x)=0 \end{cases} \tag{5}$$

Note that the marking of the place x changes according to the SAN atomic logic throughout the lifetime of the system. The transition times and marking values vary between trials. Assuming M trials, the expected value of the reward function of the place, r_x (probability for being in place x at time t), is calculated as:

$$\hat{p}_x(t) = \frac{1}{M}\sum_{i=1}^{M} r_x^i(t), x \in E \tag{6}$$

where E denotes the set of places in the system, e.g. $E = \{$`W`, `F`$\}$ in Fig. 14 and $r_x^i(t)$ denotes reward value at time t of the i-th trial. The required number of iterations M depends on the required confidence level for the reward variables.

5.3 Transformation from HiP-HOPS to SAN

The transformation process is done in two steps (cf. Fig. 7). Firstly the HiP-HOPS functional specification is transformed into the behavioural model in SAN (Subsec. 5.3.1). Then, the failure model is created according to the system structure function and it is connected with the behavioural model to propagate failure, repair, and status information events from the behavioural to the failure model (Subsec. 5.3.2).

The source model G is a HiP-HOPS State Machine (see Definition 1) and at each step of the transformation process, the modelling constructs of the model G are replaced by the equivalent SAN modelling constructs (see Definition 2) in the destination model D. The transformation steps are expressed as $LHS = RHS$, where LHS expresses the left hand side of the rule, and RHS is the right hand side of the rule. The transformation works as follows: find an occurrence of LHS in G; if found, transform LHS in RHS in the target graph D. As Table 3 displays, there is a direct correspondence ($\mathcal{C}$) between the HiP-HOPS State Machine and SAN elements.

Note that it may have been possible to express $\mathcal{C}6$ with SAN output gates. However, we use input gates for the flexibility provided for defining complex guarded expressions.

TABLE 3
Correspondence between HIP-HOPS and SAN elements.

$\mathcal{C}$	HiP-HOPS State Machine	SAN
1	States; $s_i \in S, 1 \leqslant i \leqslant N$	Places; $p_i \in P, 1 \leqslant i \leqslant N$
2	Failure/Repair events; $e_i \in \sum, \tau = \{stochastic\}$	Timed activities; $a_i \in A, \tau = \{timed\}$
3	Input propagated event; $e_{IP} \in \sum_{IP}$	Instantaneous activities; $a_i \in A; \tau = \{inst.\}$
4	Output propagated event; $e_{OP} \in \sum_{OP}$	Out propagated activity a_{OP}; out propagated place p_{OP}
5	Not occurred; occurred event	μ(place)=0; μ(place)=1
6	Conditional events; *Event[Guard]|Action events*	Input gates; IG

The HiP-HOPS event propagation is implemented by creating a shared place between the source and destination components ($\mathcal{C}4$). When the source component changes the marking of the shared place it affects the destination components that share the same place. The marking of the place must be updated every time the analysed event occurs.

5.3.1 Behavioural Model

The transformation of each component's functional model modelled with HiP-HOPS State Machines ($H2SM$, Fig. 8) into SAN behavioural model A_SAN (Fig. 15) is performed through the next steps implemented in Algorithm 1:

Step 1 (`lines 2-7`): replace states with places (`lines 5, 6`) and set the initial marking to 1 or 0 for all the places according to the initial state of the source functional specification models (`line 7`).

Step 2 (`lines 8-28`): replace events with activities according to the nature of the source events (`lines 8-9`) and link them with the corresponding places. That is, stochastic events are replaced with timed activities (`lines 11-14`); immediate events are replaced with instantaneous activities (`lines 15-21`); if the instantaneous event's name matches with any of the input propagated events name (`line 19`), create the corresponding place in the component to propagate the effect of its marking change inwards, i.e., event propagation (`lines 20, 21`). If the event is a conditional event, create the corresponding logic in SAN using IGs and places that model the events implicated in the transition events. Also add the equivalent C++ function which models the guard, i.e., if *event[guard]* then *[action]* (`lines 22-28`). Note that we unconditionally activate the enabling predicate (`line 25`), but we model the guarding condition using the input function (`line 26`).

Step 3 (`lines 29-37`): if the component propagates events outwards and it does not have a state counterpart in the SAN model (`line 31`), create the corresponding place and link it to the event that needs to be propagated (`lines 32-34`). To remove the marking of the added place, connect the added place and the place connected to the event to be propagated to an immediate transition (`lines 35-37`).

The components shown in Fig. 15 are created by applying Algorithm 1. The components A, B and S need to create an extra place to propagate the repair event from the component outwards, i.e., *A Repaired*, *B Repaired*, and *S Repaired* (cf. `line 32`). Accordingly, they are connected to an immediate activity to remove the token from these places, so that when the marking of the places connected to

Algorithm 1 Behavioural model generation

1: **function** HIPHOPS_2_SAN(H2SM)
2: get S from H2SM; ▷ set of states in the HiP-HOPS state machine model
3: let P=∅; A=∅; IG=∅
4: let $N = |S|$ ▷ number of states in the H2SM model
5: **for** each $s_i \in S, 1 \leqslant i \leqslant N$ **do** ▷ for all the states in the H2SM model
6: P ← P ∪ $\{p_i\}$ ▷ create SAN component place
7: **if** $s_i = s_0$ **then** $\mu(p_i) = 1$ **else** $\mu(p_i) = 0$ ▷ set the initial marking
8: **for** each $e \in \sum$ **do** ▷ for all the events in the H2SM model
9: let $e_{xy} = e,$ where $e_{xy} \in \Sigma | s_x \xrightarrow{e_{xy}} s_y, (s_x, s_y \in S^2)$ ▷ transitions from s_x to s_y: $s_x \xrightarrow{e_{xy}} s_y$
10: **switch** τ_{xy} **do** ▷ depending on the type of transition
11: **case** stochastic
12: $A \leftarrow A \cup \{a_{xy}\}$ where $\tau_{xy} = timed$ ▷ create a timed activity a_{xy}
13: link $p_x \rightarrow a_{xy}$ ▷ from state x
14: link $a_{xy} \rightarrow p_y$ ▷ to state y, i.e. $p_x \xrightarrow{a_{xy}} p_y$
15: **case** immediate
16: $A \leftarrow A \cup \{a_{xy}\}$ where $\tau_{xy} = instantaneous$ ▷ create an instantaneous activity
17: link $p_x \rightarrow a_{xy}$ ▷ from state x
18: link $a_{xy} \rightarrow p_y$ ▷ to state y, i.e. $p_x \xrightarrow{a_{xy}} p_y$
19: **if** $\exists e_{EP_{In}} \in EP_{In} | e_{xy} = e_{EP_{In}}$ **then** ▷ if the transition matches with any of the input propagated events
20: $P \leftarrow P \cup \{p_{EP_{In}}\}$ ▷ add a place that will propagate the event to other component
21: link $p_{EP_{In}} \rightarrow a_{xy}$ ▷ link the place to the activating activity: $\{p_x, p_{EP_{In}}\} \xrightarrow{a_{xy}} p_y$
22: **case** conditional ▷ create the IG that contains the transition logic
23: $IG \leftarrow IG \cup \{ig\}$ ▷ create an input gate, where $ig =< G, ena, f >$
24: $P \leftarrow P \cup \{p_{xy}\}$ ▷ create places for all the events involved in the conditional event logic
25: $ig \ni ena \leftarrow true$ ▷ activate the enabling predicate
26: $ig \ni f \leftarrow e_{xy}$ ▷ parse the conditional event logic e_{xy} into the IG function, i.e. $event[guard(s)]|action$
27: $A \leftarrow A \cup \{a_{ig}\}$ where $\tau_{ig} = instantaneous$ ▷ create an instantaneous activity
28: link $ig \rightarrow a_{ig}$ ▷ map input gate to activity: $IG \rightarrow A$
29: let $J = |P|$ ▷ number of created places in the SAN model
30: **for** each $e_{EP_{Out}} \in EP_{Out}$ **do**
31: **if** $\nexists p_j \in P | e_{EP_{Out}} = p_j, 1 \leqslant j \leqslant J$ **then** ▷ if there is no matching place
32: $P \leftarrow P \cup \{p_{EP_{Out}}\}$ ▷ create a place to propagate events outwards
33: let $a_{xy} = a_{p_{EP_{Out}}}$ ▷ take the activity with the name of the propagated event $p_x \xrightarrow{a_{xy}} p_y$, where $a_{xy} = a_{p_{EP_{Out}}}$
34: link $a_{xy} \rightarrow p_{EP_{Out}}$ ▷ link it to the created place $p_x \xrightarrow{a_{xy}} \{p_y, P_{EP_{Out}}\}$
35: $A \leftarrow A \cup \{a_{rem}\}$ where $\tau = instantaneous$ ▷ create removing activity to remove the token instantaneously
36: link $p_{EP_{Out}} \rightarrow a_{rem}$ ▷ link the place to the instantaneous activity
37: link $p_x \rightarrow a_{rem}$ ▷ link the source place of a_{xy} to remove the marking; $\{p_x, p_{EP_{Out}}\} \xrightarrow{a_{rem}} \emptyset$
38: **return** $P \cup A \cup IG$

this activity is 1, the token is removed from each of these places (cf. `lines 33-37`). Note that we have deliberately connected the input place of the event that needs to be propagated to an immediate activity to avoid interfering with the marking (behaviour) of the system.

Component *A* (resp. *B*) will be activated only when there is a token in the `Reconfigure` and `Standby` places simultaneously, *S* is working, and *B* (resp. *A*) has failed. The marking of the `Reconfigure` places is managed by the `Reconfiguration Logic` component, which is implemented according to the logic described in Fig. 8d using IGs (`lines 22-28`). The IGs implement the operation of S reconfiguring A and B depending on the system status.

After creating the components according to Algorithm 1, the event propagation between the behavioural components is defined by the interface connections implemented through join operators. The join operator creates a unique state linking the shared states between components. When the marking of a shared place is modified by the source component it will affect all the shared places of the destination components propagating its influence across the system. In Fig. 15 the places of the source components are represented in bold, e.g., *Reconfiguration Logic* generates reconfiguration signals `Rec-A` and `Rec-B`, which are propagated towards *A* and *B* respectively. Thanks to the matching names of places in the different components, the shared places are created automatically when joining the atomic models.

Subsequently, the behavioural model is connected with the failure model in order to evaluate the system's failure probability according to Eq. (4).

5.3.2 Failure Model

The objective of the failure model is to evaluate the failure probability of the system according to the structure function extracted in the qualitative analysis phase. To this end, it

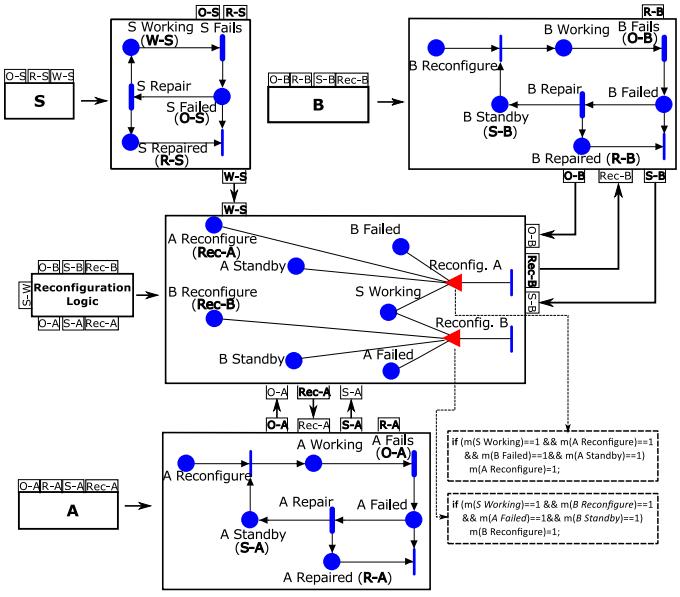


Fig. 15. Behavioural Model of the repairable PS components.

is necessary to specify the TFT gates in the target dependability evaluation formalism and connect TFT gates with the system events according to the system structure function by propagating failure and repair events from the behavioural model to the failure model (cf. Fig. 17).

For the design of the failure model SAN enables modelling the logic of the TFT gates and reusing it in order to create user-defined temporal expressions. Besides, SAN can model repeated events and subsystems through the join operator. Its modular specification enables the creation of a manageable model improving the readability and maintainability of the model for complex systems.

We focus on a subset of TFT (PAND, POR) and combinatorial (AND, OR) gates. For the systematic consideration of the gates and automatic generation of temporal expressions we introduce reusable building blocks. That is, gates are designed as functions, and the same function is used to solve all the expressions that include these gates by creating their corresponding SAN models. This is possible thanks to the modularity of SAN, which makes possible the reuse of composed and atomic models.

Instead of generalizing each TFT gate for N input events, for clarity, we characterize TFT gates for two input events (A, B). We nest operators if equations with more than two inputs are needed. Thereby, the system structure function is synthesized by connecting TFT gates with the behavioural model, and nesting the resulting composed SAN models according to the system structure function.

In SAN we design TFT gates using IGs. Places model the input events of the gates, and IGs model the gate behaviour by controlling the occurrence instances and implementing the corresponding temporal logic. The atomic models of the gates can be generated automatically or they can be imported using a library of TFT gates.

Fig. 16 shows the specification of repairable TFT gates in SAN using state machines and their corresponding SAN model. In the state machine the initial state is indicated with an arc, failure states are identified with doubled circles, and F_x and R_x indicate failure and repair events of x. The resultant reusable blocks are used to create the TFT expressions. In each of these models, the IG implements the logic expressed in the annexed code by controlling the marking of the places connected to it. The activity associated to the IG determines the nature of the IG execution. That is, it defines how the marking change will be updated by the code. In all the cases, the IGs are connected to instantaneous activities and the marking change determined by the IG is executed as soon as the logic in the code is true.

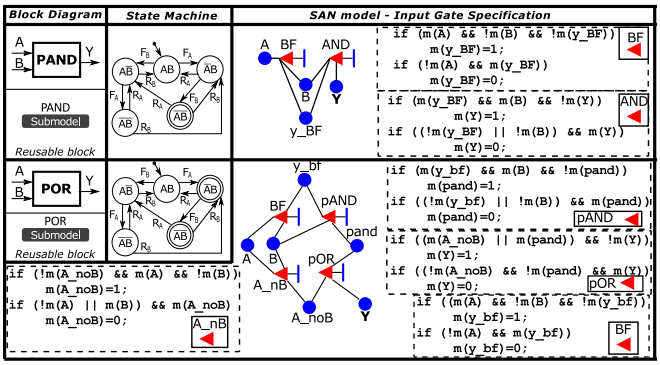


Fig. 16. Specification of TFT gates in SAN.

The PAND event happens when the first input event A happens prior to the occurrence of the second input event B. For two input events (A, B), the PAND implementation in SAN is as follows (cf. Fig. 16): A happens prior to B (`BF` input gate); and given that A happened prior to B (`y_BF` place), B happens (`AND` input gate). The resulting PAND reusable block implements the function: Y=**PAND**(A, B).

The POR event happens when the first input event `A` happens prior to the second input event `B`, independent that the event B occurs or not. For two input events, the POR implementation in SAN is as follows (Fig. 16). POR is true if either: A happens prior to B (`BF` input gate), and B occurs (`pAND` input gate); or A happens prior to B and B does not occur (i.e., `A_nB` input gate). That is, POR is true when the PAND expression is true (`m(pand)=1`, `pAND` input gate) or `A_noB` expression is true (`m(A_noB)=1`, `A_nB` input gate) as defined by the `pOR` input gate in Fig. 16. The resulting POR reusable block implements the next function: Y=**POR**(A, B).

The TFT reusable blocks can be used to solve any temporal expression which involves the AND, OR, PAND, and POR operators. To this end, places A and B will be shared with the corresponding events to be analysed and the Y place will indicate the occurrence probability of the corresponding TFT expression. Note that the TFT gate specifications in Fig. 16 are extendible to N input events. For example, it is possible to connect *N-1* 2-input TFT gates to obtain an *N*-input TFT gate. However, it is important to remember that all TFT gates are left associative [11], i.e. A|B|C is evaluated as (A|B)|C in the same way as A<B<C is evaluated as (A<B)<C.

We have defined a set of reusable models that implement the logic of the TFT and Boolean gates, i.e., library of TFT gates. Using these functions, we evaluate any MCSQ expression that contains the temporal (PAND, POR) and non-temporal (AND, OR) operators automatically by creating

composed SAN models (which include other composed and/or atomic models), that make use of the TFT gates across different composed models to solve all the temporal expressions. Fig. 17 shows the synthesis process.

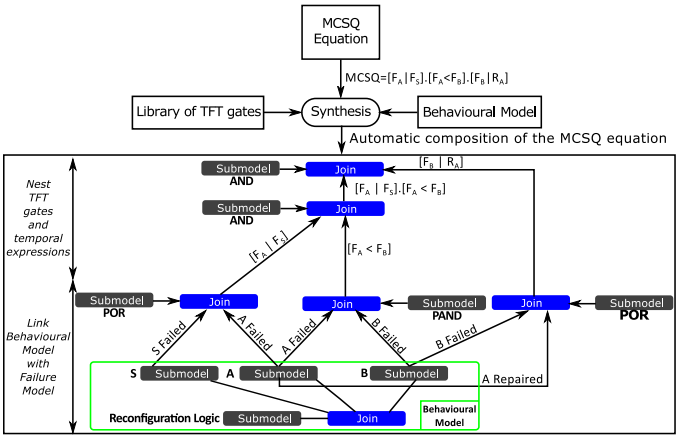


Fig. 17. Synthesis of the structure function.

The implementation of the structure function requires connecting the failure/repair events in the behavioural model with the events in the system structure function. For this, we join the corresponding SAN models by means of shared places. That is, places which correspond to the input and output events of the temporal and non-temporal operators are shared with the behavioural model's failure/repair events according to the equation that needs to be analysed. Thereby, we compose models by automatically linking the corresponding places of the behavioural model (i.e., failure and repair events) with the failure model in order to evaluate the failure probability of the system. Furthermore, replicated events and states between different expressions of the MCSQs are joined to remove repetitions in the composed SAN model.

The equation under study determines the connections between the behavioural and failure model.

For simplicity and consistency, we opt for transforming the structure function into Base Temporal Form (BTF) using doublets [11]. An expression in the BTF contains only AND gates, OR gates, and doublets. Doublets link a pair of events connected by a temporal gate indicated by square brackets. Generally it can be expressed as follows: $[In_1$ **OP** $In_2]$, where **OP** = $\{SAND, PAND, POR\}$. Doublets encapsulate the temporal information in an expression so that it can be manipulated like a non-temporal expression.

In BTF, the resulting structure function will be a disjunction of individual MCSQ sets and each set will have doublets, normal events, or combinations thereof linked by AND gates. Eq. (7) shows the structure function in BTF.

$$TE = \sum_{i=1}^{M} MCSQ_i; MCSQ_i = \prod_{j=1}^{K} event_{ij} \tag{7}$$

where $event_{ij} \in \{doublet, event\}$; *doublet* = [$event_x$ **OP** $event_y$] and *event* is any failure/repair event characterized with its corresponding PDF.

Any expression containing temporal gates can be converted into a set of doublets via the temporal laws introduced in [11]. Accordingly, Eq. (4) is transformed into:

$$\begin{aligned} MCSQ_1 &= [F_A < F_B].[F_A|F_S].[F_B|R_A] \\ MCSQ_2 &= [F_B < F_A].[F_B|F_S].[F_A|R_B] \\ MCSQ_3 &= [F_S < F_A].[F_S|F_B].[F_A|R_S] \\ MCSQ_4 &= \\ &[R_A < F_B].[F_S < F_B].[F_B < R_S].([F_A < F_S] + [F_A < R_A]) \end{aligned} \tag{8}$$

$MCSQ_1$, $MCSQ_2$ and $MCSQ_3$ are created directly by applying the temporal equivalence $ID6$ in Table 1 and $MCSQ_4$ is obtained through applying sequentially $ID5$, $ID2$, $ID3$, $ID6$, and $ID1$ temporal laws in Table 1 to $MCSQ_4$.

Algorithm 2 generates automatically composed SAN models according to the structure in Eq. (4), taking as input the structure function (TE) and the behavioural model of the system. The algorithm iterates among the M disjunctive MCSQ expressions. For each event in the MCSQ expression it is checked if it is a doublet or not (`line 7`). If it is, first the left hand side input, right hand side input, and the logical operation of the doublet are extracted through LHS, RHS, and gate functions, respectively (`lines 8-10`). Then the doublet operation is transformed into the corresponding SAN composed model using the *join* function. The join function links input events specified in the *Model* with the corresponding operator creating a composed model (`line 11`). To this end, the $join(OP, In_1, In_2, Model)$ function will search in the $Model$ the places In_1 and In_2 (identified by their names) and then it will join them with the operation OP (In_1 with the first input of OP, and In_2 with the second input of OP) creating a SAN composed model.

If there are more than one conjunctive expressions in the equation $MCSQ_i \in TE$, these are nested iteratively by connecting them using the SAN composed models via the function *nest* (`line 17`). The function $nest(OP, M_1, M_2)$ links the first input of the operation OP with the output of the composed SAN model M_1 and the second input of the operation OP with the output of the composed SAN model M_2 (Fig. 17). Once the inner for loop is over (`line 18`) the variable $SAN\_composed\_MCSQ_i$ will specify a MCSQ expression in a SAN composed model which will have inner composed models if the expression has conjunctive events.

Depending on the number of MCSQ expressions in the structure function (`line 20`), MCSQ expressions are nested iteratively using OR gates via the `nest` function that will link composed models with TFT gates using shared places (`line 23`). The variable $SAN\_composed\_TE_i$ will include all the information corresponding to the top event occurrence by nesting composed and atomic SAN models. Finally, places with the same name are shared automatically in the resulting SAN model that implements the structure function (`line 25`). The failure probability is then quantified by Monte Carlo simulations as defined in Eq. (6).

5.4 Repairable PS System: Results

The generic functionality of the repairable PS system can be seen as a power supply system which has the primary power supply (A), the secondary power supply (B), the fault detection and reconfiguration component (S) and the input power (I). Plausible values are assigned to each component. Without loss of generality, all the system events are characterized with exponential distributions according to their corresponding failure and repair rates (all rates in hours^{-1}

Algorithm 2 SAN model generation from MCSQ expressions

function SYNTHESIZE_TE_SAN(TE, BehaviouralModel)
  let $MCSQ_i \in TE, 1 \leqslant i \leqslant M$; ▷ M disjoint sets
  for each $MCSQ_i \in TE$ **do** ▷ transform BDMP elements into SAN
    let $len_{MCSQ} = |MCSQ_i|$ ▷ number of doublets or events in $MCSQ_i$, i.e., AND-ed conjunctive terms
    let $event_{ij} \in MCSQ_i, 1 \leqslant j \leqslant len_{MCSQ}$ ▷ doublet(s) or event(s) in $MCSQ_i$
    for each $event_{ij} \in MCSQ_i$ **do**
      if $event_{ij}$=$doublet$ **then** ▷ check if it is a doublet
        let $Op = gate(event_{ij})$ ▷ Operation, e.g. $A < B$
        let $In_1 = LHS(event_{ij})$ ▷ LHS of the doublet, e.g. $\mathbf{A} < B$
        let $In_2 = RHS(event_{ij})$ ▷ RHS of the doublet, e.g. $A < \mathbf{B}$
        let $SAN\_composed_j = join(Op, In_1, In_2, BehaviouralModel)$ ▷ Create SAN composed model
      else let $SAN\_composed_j = event_{ij}$ ▷ Non-temporal single event
      if j=1 **then** ▷ $event_{ij} \in MCSQ_i; MCSQ_i \in TE$
        if $len_{MCSQ} = 1$ **then** let $SAN\_composed\_MCSQ_i$=$SAN\_composed_j$ ▷ single event no linking logic
        else let $SAN\_composed\_nest$=$SAN\_composed_j$ ▷ variable to store nested events
      else ▷ Nest in SAN, AND-ing previous events in a SAN composed model
        let $SAN\_composed\_MCSQ_i$=$nest(AND, SAN\_composed\_nest, SAN\_composed_j)$
        let $SAN\_composed\_nest$=$SAN\_composed\_MCSQ_i$ ▷ prepare for the next iteration
    if i=1 **then** ▷ $MCSQ_i \in TE$
      if $M = 1$ **then** let $SAN\_composed\_TE_i$=$SAN\_composed\_MCSQ_i$ ▷ A single MCSQ term
      else let $SAN\_composed\_TE\_nest$=$SAN\_composed\_MCSQ_i$
    else ▷ Nest disjoint sets by using OR gates
      let $SAN\_composed\_TE_i$=$nest(OR, SAN\_composed\_TE\_nest, SAN\_composed\_MCSQ_i)$
      let $SAN\_composed\_TE\_nest$=$SAN\_composed\_TE_i$ ▷ prepare for the next iteration
  return $share\_common\_places(SAN\_composed\_TE_i)$

units): λ_I=1e-4, μ_I=2.5e-1; λ_A=2.3e-3, μ_A=2.6; λ_B=5.3e-3, μ_B=8.7e-2; λ_S=1e-4, μ_S=2.5e-1.

Three configurations have been analysed focusing on the properties of the approaches addressed in Section 4:

#1 Repairable HiP-HOPS (behavioural model according to Fig. 10 and failure model according to Eq. (4)).
#2 SEFT approach (behavioural model according to Fig. 5 and failure model according to Eq. (1)).
#3 State-of-the-art approaches (behavioural model according to Fig. 6 and failure model according to Eq. (1)).

The behavioural model and failure model of each configuration depends on the underlying assumptions of each configuration. As for the behavioural model, the components shown in Fig. 15 describe component operations for the configurations #1 and #2. With configuration #3 the behaviour of the components A, B, and Reconfiguration Logic is different: A does not have a standby state and, according to the Reconfiguration Logic, when A is repaired B returns back to the standby state. Furthermore, in order to model the state transition from state #4 to #3 in the FA shown in Fig. 6, the status of S is not checked when performing the reconfigurations. With respect to the failure model the configuration #1 implements Eq. (4), and configurations #2 and #3 implement Eq. (7). Fig. 18 shows the failure probability values for these configurations at different time instants. All the simulations have been performed with confidence level = 0.99 and confidence interval = 9e-6.

The differences between configuration #1 and configurations #2 and #3 are caused by $MCSQ_4$, which is not covered by the state-of-the-art approaches in general, and configurations #2 and #3 in particular. With respect to configuration

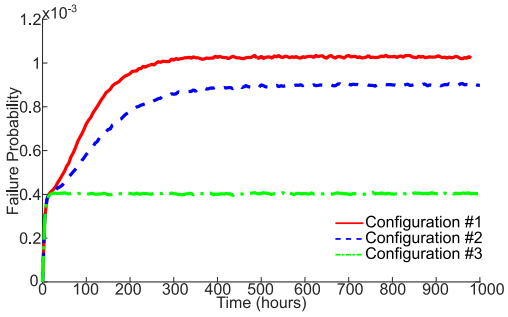


Fig. 18. Failure probability of the tested configurations.

#3 we can see that the differences are higher. This is because the modelled system behaviour in configuration #3 does not match with the desired behaviour of the primary standby system and results are thus based on different underlying Failure Automaton. The obtained failure probability values may lead the designer to adopt optimistic design decisions for this scenario which is potentially problematic.

6 DISCUSSION

The computational and modelling complexity of the analysis can increase substantially with system scale. This can be a substantial issue so care must be taken in the scope of application of the techniques described in the paper.

SAN models can be directly solved via Monte Carlo simulations, which can be parallelized so as to speed-up complex system simulations. In addition, in order to alleviate substantially the state-explosion problem, SAN makes use of reduced base models [24]. This concept enables the

implementation of join operators and hierarchical modelling of complex systems (see Section 5.2).

The modelling effort, e.g. the specification of failure automata for each component, can be a cumbersome process. However, this is a trade-off decision that the designer needs to make when designing dynamic repairable systems: either focus on simpler and more coarse approaches with fixed predefined constructs for representing repair or follow a more detailed modelling process to address accurately all the possible failure and repair scenarios.

One possible solution to alleviate the modelling effort may be to automate the model transformation steps. The system architecture specification and component annotations need to be manually defined (cf. Fig. 7) because this is dependent on expert knowledge. However, it is possible to automate subsequent qualitative and quantitative analysis steps. Indeed, as indicated in the next section, this is part of our future work.

7 Conclusions

We presented a novel approach for analysis of dynamic repairable systems. Compared with the state-of-the-art, the proposed approach differs in that repair processes are explicitly modelled. Analyses evaluate the influence of failure and repair scenarios under assumptions of any PDF. The proposed approach captures a detailed view of the effects of failure and repair and can give improved insights which are currently unavailable. The particular implementation of the approach via HiP-HOPS and SAN brings the computational combined benefits of these tools. However, the key idea of explicit treatment of repair in dependability analysis could also inform the development of other techniques. Applicability of this concept is important in systems where the possibilities of repair multiply with the ability to dynamically redirect resources via software reconfiguration.

Some of the disadvantages of Petri Net models including modelling complexity and state explosion are to some extent inherited in this approach. However, the hierarchical structure of SAN models and their underlying resolution method mean that to some extent these issues are addressed. SAN models are less computationally expensive to solve, less error-prone and less difficult to maintain than classical Petri Net models.

The different formalisms and models used in this work have been designed to demonstrate the feasibility and added value of the proposed approach. In principle other reliability modelling and solving tools like Möbius could be connected to HIP-HOPS and Pandora to enable similar capabilities for analysis of systems, but that would require substantial conceptual work and tool extensions. Future work may consider the integration of Pandora, TFTs, SMs, FA, and SAN models in an integrated automated software tool within the HiP-HOPS framework.

Acknowledgment

This work was partially supported by the DEIS H2020 project (Grant Agreement 732242).

References

[1] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, 2004.
[2] IEC, "Analysis Techniques for Dependability," *IEC 61078*, 2006.
[3] IEC, "Fault Tree Analysis," *IEC 61025*, 2006.
[4] G. Manno, F. Chiacchio, L. Compagno, D. D'Urso, and N. Trapani, "Conception of Repairable Dynamic Fault Trees and resolution by the use of RAATSS, a Matlab toolbox based on the ATS formalism," *Rel. Eng. & Sys. Safety*, vol. 121, pp. 250 – 262, 2014.
[5] S. Distefano and A. Puliafito, "Dependability evaluation with dynamic reliability block diagrams and dynamic fault trees," *IEEE Trans. Dependable Secure Comput.*, vol. 6, no. 1, pp. 4–17, Jan 2009.
[6] IEC, "Functional safety of electrical/electronic/programmable electronic safety related systems," *IEC 61508*, 2000.
[7] Z. Tang and J. Bechta Dugan, "Minimal cut set/sequence generation for dynamic fault trees," in *Reliability and Maintainability, 2004 Annual Symposium - RAMS*, Jan 2004, pp. 207–213.
[8] J. B. Dugan, S. J. Bavuso, and M. A. Boyd, "Dynamic fault-tree models for fault-tolerant computer systems," *IEEE Transactions on Reliability*, vol. 41, no. 3, pp. 363–377, Sep 1992.
[9] P.-Y. Chaux, J.-M. Rousel, J.-J. Lesage, G. Delezeu, and M. Bouissou, "Towards a Unified Definition of Minimal Cut Sequences," in *Proc. of Dependable Control of Discrete Systems*, vol. 4, 2013, pp. 1–6.
[10] Y. Papadopoulos, M. Walker, D. Parker, S. Sharvia, L. Bottaci, S. Kabir, L. Azevedo, and I. Sorokos, "A synthesis of logic and bio-inspired techniques in the design of dependable systems," *Annual Reviews in Control*, vol. 41, pp. 170 – 182, 2016.
[11] M. Walker and Y. Papadopoulos, "Qualitative temporal analysis: Towards a full implementation of the fault tree handbook," *Control Engineering Practice*, vol. 17, no. 10, pp. 1115 – 1125, 2009.
[12] G. Merle, J. Roussel, J. Lesage, and A. Bobbio, "Probabilistic algebraic analysis of fault trees with priority dynamic gates and repeated events," *IEEE Trans. Rel.*, vol. 59, no. 1, pp. 250–261, 2010.
[13] W. H. Sanders and J. F. Meyer, "Stochastic activity networks: Formal definitions and concepts," in *Lectures on Formal Methods and Performance Analysis*. Springer, 2001, vol. 2090, pp. 315–343.
[14] D. Codetta-Raiteri, "Integrating several formalisms in order to increase fault trees' modeling power," *Reliability Engineering & System Safety*, vol. 96, no. 5, pp. 534 – 544, 2011.
[15] M. Bouissou and J. Bon, "A new formalism that combines advantages of fault-trees and markov models: Boolean logic driven markov processes," *Rel. Eng. & Sys. Safety*, vol. 82, no. 2, pp. 149 – 163, 2003.
[16] B. Kaiser, C. Gramlich, and M. Förster, "State/event fault treesa safety analysis model for software-controlled systems," *Reliability Engineering & System Safety*, vol. 92, no. 11, pp. 1521 – 1537, 2007.
[17] P.-Y. Piriou, J.-M. Faure, and J.-J. Lesage, "Generalized boolean logic driven markov processes: A powerful modeling framework for model-based safety analysis of dynamic repairable and reconfigurable systems," *Reliability Engineering & System Safety*, vol. 163, pp. 57 – 68, 2017.
[18] J. I. Aizpurua and E. Muxika, "Model based design of dependable systems: Limitations and evolution of analysis and verification approaches," *Int. J. on Advances in Security*, vol. 6, pp. 12–31, 2013.
[19] N. Mahmud, Y. Papadopoulos, and M. Walker, "A translation of state machines to temporal fault trees," in *Int. Conf. on Dependable Systems and Networks*, June 2010, pp. 45–51.
[20] D. Coppit, K. J. Sullivan, and J. B. Dugan, "Formal semantics of models for computational engineering: a case study on dynamic fault trees," in *Proc. of IEEE ISSRE*, 2000, pp. 270–282.
[21] G. Manno, A. Zymaris, N. Kakalis, F. Chiacchio, F. Cipollone, L. Compagno, D. D'Urso, and N. Trapani, "Dynamic reliability analysis of three nonlinear aging components with different failure modes characteristics," *Safety, Reliability and Risk Analysis: Beyond the Horizon*, pp. 3047–3055, 2013.
[22] D. D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster, "The mobius framework and its implementation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 956–969, Oct 2002.
[23] C. Di Martino, M. Cinque, and D. Cotroneo, "Automated generation of performance and dependability models for the assessment of wireless sensor networks," *Computers, IEEE Transactions on*, vol. 61, no. 6, pp. 870–884, June 2012.
[24] W. H. Sanders and J. F. Meyer, "Reduced base model construction methods for stochastic activity networks," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 1, pp. 25–36, Jan 1991.

Jose Ignacio Aizpurua (M'17) is a Research Associate within the Institute for Energy and Environment at the University of Strathclyde, Scotland, UK. He received his Eng., M.Sc., and Ph.D. degrees from Mondragon University (Spain) in 2010, 2012, and 2015 respectively. He was a visiting researcher in the Dependable Systems Research group at the University of Hull in 2014. His research interests include prognostics and health management, dependability, condition monitoring and systems engineering.

Yiannis Papadopoulos is a professor and leader of the Dependable Intelligent Systems research group at the University of Hull. He pioneered the HiP-HOPS MBSA method and contributed to the EAST-ADL automotive design language, working with Volvo, Honda, Continental, Honeywell, and DNV-GL, among others. He is actively involved in two technical committees of IFAC (TC 1.3 & 5.1).

Guillaume Merle received the Ph.D. degree from the Ecole Normale Supérieure de Cachan (France) in 2010. He is currently Professeur Agregé of engineering science at Beihang Sino-French Engineering School (Beijing, China), where he teaches engineering science, automation control and engineering thermodynamics.